Desk-Top Software Development Using HTML Applications

John N. Dyer, Department of Information Systems, College of IT, Georgia Southern University, P.O. Box 7998, Statesboro, GA 30459

Abstract

While many business managers today have experience with office productivity software, internet browsing, and html development, few outside of the I.T. field have the knowledge, skills and experience in developing desk-top applications. The purpose of this article is to provide a relatively simple (but little known) method of developing and deploying Windows client-side applications, called html applications (HTA), using html, Windows Scripting Languages (WSL), and Windows COM and Objects. Few standard programming courses (with the exception of Visual Basic (VB)) include topics directly related to Windows application development. A recent literature review found no textbooks or journal articles, and few websites referencing the use of HTA in the development of Windows applications. The goal of this article is to increase the manager's programming knowledge base, thus adding greater value to applications development, regardless of the skill of the programmer. As such, this article provides an extensive overview of HTA development, including a description of HTA and related components, terminology, source references, example applications, sample code, and screen shots of working HTAs.

Keywords
HTA, HTML, Scripting, Programming, Client-Side Applications

Introduction

In laymen's terms, an HTA is a software application in which basic programming is contained inside a standard HTML web page, and converted to an HTA using the .hta file extension. In essence, an HTA is an easily programmable web page that allows development of desk-top, applications that can interact with both the operating system and other desk-top applications. Similar to software developed with Visual Basic (VB), HTAs are developed as true Windows applications, programmable with basic web scripting languages (Jscript, VB Script, JavaScript), and interacting with the

Window's Component Object Model (COM) and ActiveX technology.

Additionally, the HTA's user interface is presented using a GUI style browser design that allows all the features and tools of standard Window's software. As such, both seasoned and amateur programmers have the ability to build full-bodied applications that operate seamlessly with the Windows OS.

An HTA is also a fully trusted application complete with client-side read/write access to the OS file system, access to system objects and other applications, e.g. Excel, Notepad, etc, and the system registry. An HTA can run embedded Microsoft ActiveX controls and Java applets and interact with the OS with no warning displays before such objects are run within the HTA. (Introduction to HTML Applications (HTAs) (n.d.)).

HTA Applicability

HTA programming is applicable for simple applications or full-scale Windows applications. HTA can include forms, multimedia, Web applications, browsers, and much more. (Introduction to HTML Applications (HTAs) (n.d.)). The implementation of HTA is limited to Windows-based computers running IE 5 and later. Programming languages like C++ and VB are typically used for Window's application software, since the languages have access to Windows system resources and the object within the Windows OS. With HTAs, HTML with script can be used as an alternative to higher level programming languages, with the same access to Windows system resources. Just as a web browser, HTAs support HTML, Cascading Style Sheets (CSS), scripting languages, and also include HTA-specific functionality, thus providing control over user interface design and access to the OS. The HTA runs like any executable (.exe) written in C++ or VB.

Other features of HTA that make it suitable for client-side applications (Introduction to HTML Applications (HTAs) (n.d.)) include OS file manipulation capabilities, Windows Scripting Host support, database access, access to Microsoft Office applications, and FTP.

HTA CODING APPLICABILITY

Although an HTA can be written in a text editor, it is suggested that an editor similar to one used to create Web pages. Suitable editors include Microsoft FrontPage and Visual Studio. Since the HTA application is script wrapped in

HTML elements, it is suggested that the reader review the HTML tutorial available online (W3Schools (n.d.)), and in particular become familiar with the HTML Document Object Model (HTML DOM (n.d.)).

HTA PROGRAMMING WITH BASIC JAVASCRIPT

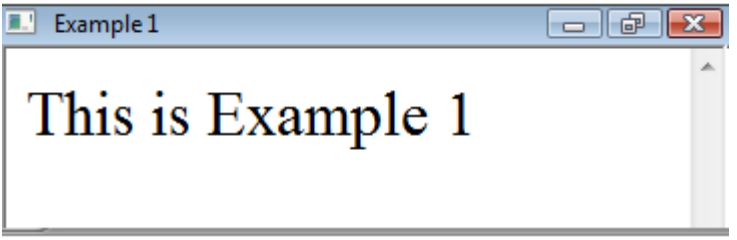
An HTA consists of three sections, that together provide the desktop application. The sections include the head section, the script section and the body section. The head section consists of the standard HTML tags, <html> and <head>. To make the file run as an HTA, an additional tag must be included inside the <head> tag, specifically, <HTA: APPLICATION>, and the file must be saved with the .hta extension. Where the .hta extension tells the operating system how to run the application, the HTA tag tells the window how to behave as an application. This tag also exposes a limited set of HTA specific attributes that control everything from border style to a program icon and a menu. Most attributes have default values optimized for the average application and need not be specified. The available attributes and properties specific to the tag are shown in Appendix A. The following example shows the required HTA head, script (using JavaScript) and body sections, with no attributes or properties set for the HTA tag.

<html><head><HTA: APPLICATION><title> *optional* </title></head>
<script type="text/javascript"> *script goes here* </script>
<body> *html elements go here* </body>
</html>

An HTA is executed by double-clicking the program icon, running it from the Windows Start menu, opening it through a URL, or starting it from the Windows command line (Run). When executed, the HTA window title bar displays any text contained in the title tag, renders html elements and text contained in the body, and executes scripts automatically or by user interaction.

Example 1: Simple Text Display
Example 1 shows the markup of a file saved as Example1.hta, while Figure 1.1 shows a screen-shot of the HTA. After executing the HTA, the result is a window with “Example 1” displayed in the blue title bar at top beside a default system icon, the text “This is Example 1” displayed in the application active window, and the usual navigation buttons displayed on the right end of the title bar used for minimizing, restoring, and closing the application. The keyboard combination ALT+F4 will also close the application.

Figure 1.1



Code: Simple Text Display

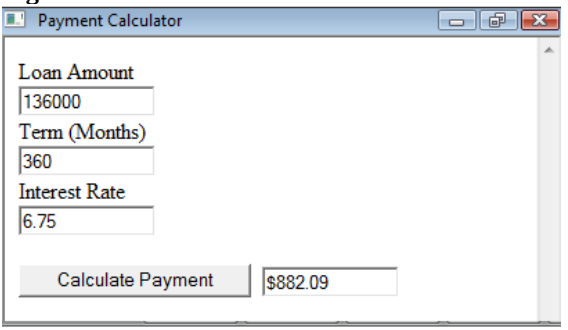
```
<html><head><HTA: APPLICATION><title> Example 1 </title></head>
<script type="text/javascript"></script>
<body> This is Example 1</body>
</html>
```

Example 1 is limited in scope but shows the required structure for an HTA. The subsequent examples provide the code and code comments for various HTA programs involving html elements and controls, scripting, and COM interaction.

Example 2: Loan Payment Calculator

Figure 2.1 displays an HTA loan payment calculator, using JavaScript to calculate and display results. The user enters the loan amount, loan term, and interest rate in the text boxes, then clicks the "Calculate Payment." The monthly payment is then calculated and displayed in a text box. The code below reflects the markup for the required sections. The code comments are also provided. After writing the markup it should be saved as "Payment.hta."

Figure 2.1



Code: Loan Payment Calculator

```
1       <html><head><HTA: APPLICATION><title>Payment
Calculator</title></head>
2       <body>
3       <form name="frmInput">
4       Loan Amount<br>
5       <input type="text" name="txtAmount" size="12"> <br>
6       Term (Months) <br>
7       <input type="text" name="txtTerm" size="12"> <br>
8       Interest Rate <br>
9       <input type="text" name="txtRate" size="12"> <br> <br>
10      <input type="button" value="Calculate Payment"
        name="btnCalculate"
         onclick="calcPayment()”>
11      <input type="text" name="txtPmt" size="12">
12      </form>
13      <SCRIPT type="text/javascript">
14      function calcPayment()
15      {
16      varAmount = document.frmInput.txtAmount.value;
17      varTerm = document.frmInput.txtTerm.value;
18      varRate = document.frmInput.txtRate.value;
19      varAdjRate = (varRate/12)/100;
20      varNumCalc =varAdjRate* Math.pow((1+varAdjRate),varTerm);
21      varDenomCalc= Math.pow((1+varAdjRate),varTerm)-1;
22      varPayment = (varAmount*varNumCalc/varDenomCalc).toFixed(2);
23      document.frmInput.txtPmt.value = "$"+varPayment;
24      }
25      </SCRIPT>
26      </body></html>
```

Code Comments: Loan Payment Calculator

1 Head section opening tags with HTA tag and optional title tag, enclosing the application title.
2 Body section opening tag.
3 Form element opening tag with optional form *name attribute* set to "frmInput."
4 Text "Loan Amount" displayed in the application window, and a line break.
5 Input element (text box) tag for loan amount, with *name attribute* set to "txtAmount" and *size attribute* set to "12", and a line break.
6 Text "Term (Months)" displayed in the application window, and a line break
7 Input element (text box) tag for loan term, with *name attribute* set to "txtTerm" and *size attribute* set to "12", and a line break.
8 "Interest Rate" displayed in the application window, and a line break.
9 Input element (text box) tag for loan interest rate, with *name attribute* set to "txtRate" and *size attribute* set to "12", and a line break.
10 Input element (button) tag for button control, with *value attribute* (button text) set to "Calculate Payment," *name attribute* set to "btnCalculate," and *onClick attribute* set to the function name "calcPayment()," which binds the function to the button control.
11 Input element (text box) tag for loan payment, with *name attribute* set to "txtPmt" and *size attribute* set to "12."
12 Form element closing tag.
13 Script section opening tag with *type attribute* set to "text/javascript."
14 Function declaration "function" and required function name "calcPayment()."
15 JavaScript character "{", indicating the opening of script.
16 Variable declaration for "varAmount" set equal to the value entered by user into the "txtAmount" textbox control (line 5). The value is assigned using the HTML DOM Object.Property.Method specification, document.Object(FormName).Object(ControlName).Property(value) Note that each line of script is ended with the character ";"
17 Variable declaration for "varTerm" set equal to the value entered by user into the "txtTerm" textbox control (line 7).
18 Variable declaration for "varRate" set equal to the value entered by user into the "txtRate" text box control (line 9).
19 Variable declaration for "varAdjRate" set equal to the constant calculated as (varRate/12)/100;
20 declaration for "varNumCalc" set equal to the numerator of the payment equation (Appendix xx). Note the use of the JavaScript Math Object "Math.pow"

21 Variable declaration for "varDenomCalc" set equal to the denominator of the payment equation.
22 Variable declaration for "varPayment" set equal to a function of previous variables of the payment equation. Note the use of the JavaScript Number Object "to.Fixed" used to convert the calculated payment value to two decimal places.
23 Sets and displays the value of "varPayment" in the textbox control (line 11) using the HTML DOM specification for data binding. Note the "$"+, that concatenates (+) the dollar symbol "$" with the payment value variable.
24 Required JavaScript character "}", indicating the end of script.
25 Script section closing tag.
26 Body and Head section closing tags</body></html>

HTA PROGRAMMING USING FOS, COM AND ACTIVEX OBJECTS

The true value of an HTA is its ability to interact with COM technology (What is COM? (n.d.)), (COM Objects (n.d.). Within the Windows OS, COM allows software applications to communicate, and enables an application to use system objects. COM is often used by developers to link components together to build applications, and to take advantage of Windows OS services. For example, COM OLE technology allows applications to manipulate Microsoft Excel and other Office applications, and COM Automation allows users to build scripts in their applications to control one application from another. COM technologies also include ActiveX Controls.

The COM platform provides a standard way for an HTA to make system objects available to any COM-compliant application. In laymen's terms, COM makes it possible for nonprogrammers to write applications within the Windows OS. COM provides a translation of simple scripts into commands that can be acted on by the OS. Effectively, COM allows Windows Desk-top programming for those who have no high-level programming experience.

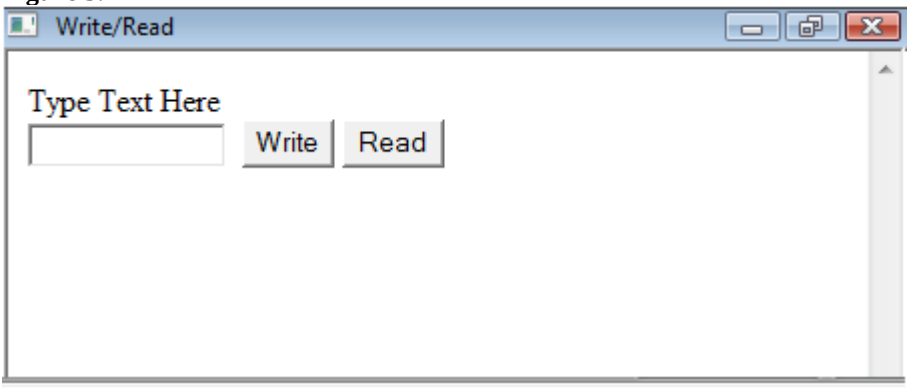
COM components are files that contain definitions of the objects the components have available for use. When a COM object is created in a HTA script a copy of one of the classes contained within the COM component is created. After the instance has been created, the HTA can take advantage of the properties, methods, and events exposed by the object.

Scripting languages work with a large subset of objects known as Automation objects, but not all. One important subset of COM is ActiveX, which consists of objects representing a particular function, a set of functions, and control objects that can be used to create HTAs that work over the Internet through web browsers. ActiveX components are only compatible with IE and the Windows OS. ActiveX controls are granted a much higher level of control over the Windows OS than Java applets. An HTA can use both ActiveX components and Java applets.

Example 3: Write/Read Application

An important object displayed in Example 3 is the FileSystemObject (FSO), which allows use of the *object.property.method* syntax with a large set of properties, methods, and events used to gain read/write access to system folders and files. The FSO exposes the objects necessary to add, move, change, create, or delete folders (directories) and files on the client computer. It also enables retrieval of information and manipulation of client drives (FileSystemObject Sample Code (n.d.)). Example 3 demonstrates using the FileSystemObject (FSO) COM (FileSystemObject Basics (n.d.)) to write to and read from the OS file system. Figure 3.1 is a screen-shot of the HTA.

Figure 3.1



When user types text into the textbox and clicks the "Write" button, the HTA creates, names and writes the user's text into a text file on the desk-top host. When the "Read" button is clicked the text file is opened and the contents are displayed in the HTA window. Both operations result in an alert box relating a custom message to the user, as shown in Figures 3.2 and 3.3.

Figure 3.2

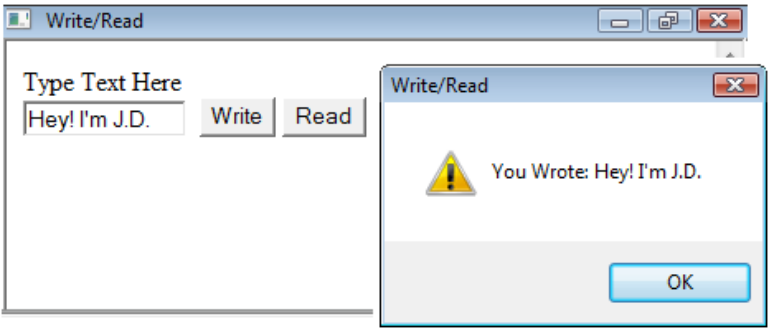
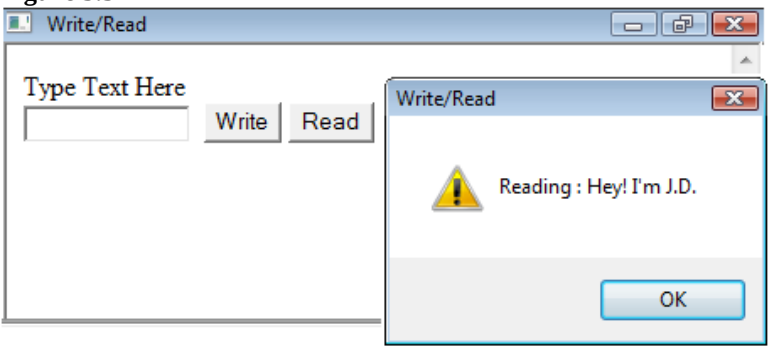


Figure 3.3



Although the complete code markup is provided the subsequent code comments relate only to the two functions whose script manipulates the FSO COM, an ActiveX COM, and several system objects.

Code: Write/Read Application

```
1
        <html><head><HTA:APPLICATION><title>Write/Read</title></h
ead>
2       <script type="text/javascript">
3       function WriteNewFile()
4       {
5       var myText = document.frmInput.txtInput.value;
6       var newFSO, cMyFile, myFileName;
7       myFileName = "myText.txt";
8       newFSO = new ActiveXObject("Scripting.FileSystemObject");
9       cMyFile = newFSO.CreateTextFile(myFileName, true);
```

```
10      cMyFile.WriteLine(myText);
11      cMyFile.Close();
12      alert("You Wrote: "+myText);
13      }
14      function ReadNewFile()
15      {
16      var newFSO, oMyFile, myFileName;
17      myFileName = "myText.txt";
18      ForReading = 1;
19      newFSO = new ActiveXObject("Scripting.FileSystemObject");
20      oMyFile = newFSO.OpenTextFile (myFileName, ForReading);
21      alert("Reading : "+oMyFile.ReadLine());
22      oMyFile.Close();
23      }
24      </script>
25      <body><form name="frmInput">Type Text Here<br>
26      <input type="text" name="txtInput" size="12">
27      <input type="submit" value="Write" name="btnWrite"
        onclick="WriteNewFile()">
28      <input type="submit" value="Read" name="btnRead"
        onclick="ReadNewFile()">
29      </form></body></html>
```

Code Comments: Write/Read Application

5 Variable declaration for “myText” set equal to the value entered by user into the “txtInput” textbox control (line 26).
6 Variable declarations for object holders.
7 Sets variable “myFileName” equal to "myText.txt", which is the name of the text file.
8 Creates a new *ActiveX Object* of the "Scripting.FileSystemObject" type and assigns it to variable “newFSO.”
9 Uses the “*CreateTextFile*” method to physically create a text file with the name assigned to variable “myFileName,” and hold it in the variable “cMyFile.”
10 Uses the “*WriteLine*” method to write the text string assigned to variable “myText.”
11 Uses the “*Close*” method to close the text file.
12 Uses the system “*alert*” object to write a screen message concatenating the text string "You Wrote: " with the variable “myText.”

…..

16 Variable declarations for object holders.
17 Sets variable “myFileName” equal to "myText.txt", which is the name of the text file.

18 Declares a variable named “ForReading” and sets it to the value 1, which is a required property value of the “*OpenTextFile*” method.
19 Creates a new *ActiveX Object* of the "Scripting.FileSystemObject" type and assigns it to variable “newFSO.”
20 Uses the “*OpenTextFile*” method to open the text file for reading.
21 Uses the system “*alert*” object to write a screen message concatenating the text string "Reading: " with the contents of the text file returned using the “*ReadLine*()” method.
22 Uses the “*Close*” method to close the text file.

Example 4: Excel Loan Amortization

Examples 4 used an ActiveXObject to interact with Excel, an Automation Object. Figure 4 is a screen-shot of the HTA. The HTA calculates of a complete loan amortization. When the HTA is executed, it automatically opens a hidden instance of Excel 2007, binds to the built-in loan amortization template, and binds default values for the first five text boxes in the first column. The path for the template is typically located in C:/Program Files (x86)/Microsoft Office/Templates/1033/LoanAmortization.xltx.

The user clicks the “Calculate Amortization” button to complete the amortization; the output from the Excel template is bound to the HTA and displayed. The user can also print the results, reset the amortization, enter new inputs, and run as many times as desired. The HTA also checks for the appropriate versions of Excel and I.E. and warns if the incorrect versions are detected. Although the complete code markup is provided, no comments are provided. The HTA embeds the following four function scripts.

1. versionCheck() – validates the correct versions of Excel and I.E. when HTA is loaded.
2. showAmort() – opens the Excel template and binds default values to the appropriate cells; completes amortization using the “Calculate Amortization” button.
3. resetPage() – resets the amortization schedule using the “Reset” button.
4. printPage() – prints the results using the “Print Page” button.

Figure 4

Loan Amount	150000	Scheduled Payment	$12944.46
Annual Interest Rate	6.5	Scheduled Number of Payments	12
Loan Period in Years	1	Actual Number of Payments	12
Payments Per Year	12	Total Early Payments	$0.00
Loan Starting Date	01/01/2010	Total Interest Paid	$5333.55
Optional Extra Payments	0	Calculate Amortization / Reset / Print Page	

Payment Number	Paymemt Date	Beginning Balance	Scheduled Payment	Extra Payment	Total Payment	Amount to Principal	Amount to Interest	Ending Balance
1	Feb 1 2010	150000.00	12944.46	0.00	12944.46	12131.96	812.50	137868.04
2	Mar 1 2010	137868.04	12944.46	0.00	12944.46	12197.68	746.79	125670.36
3	Apr 1 2010	125670.36	12944.46	0.00	12944.46	12263.75	680.71	113406.61
4	May 1 2010	113406.61	12944.46	0.00	12944.46	12330.18	614.29	101076.44
5	Jun 1 2010	101076.44	12944.46	0.00	12944.46	12396.97	547.50	88679.47
6	Jul 1 2010	88679.47	12944.46	0.00	12944.46	12464.12	480.35	76215.35
7	Aug 1 2010	76215.35	12944.46	0.00	12944.46	12531.63	412.83	63683.73
8	Sep 1 2010	63683.73	12944.46	0.00	12944.46	12599.51	344.95	51084.22
9	Oct 1 2010	51084.22	12944.46	0.00	12944.46	12667.76	276.71	38416.46
10	Nov 1 2010	38416.46	12944.46	0.00	12944.46	12736.37	208.09	25680.09
11	Dec 1 2010	25680.09	12944.46	0.00	12944.46	12805.36	139.10	12874.72
12	Jan 1 2011	12874.72	12944.46	0.00	12874.72	12804.99	69.74	0.00

Code: Excel Loan Amortization

```
<html><head><HTA:APPLICATION>
<title>Excel Loan Amortization</title></head>
<body onload="versionCheck()">
<div align="center">
<table border="1" width="80%" bordercolor="#000080" id="tableLoanSummary">
<tr id="T1Row1" >
        <td id="cellTextLoanAmount" width="200">Loan Amount</td>
        <td id="cellInputTextboxLoanAmount" width="75"><input type="text"
        name="textboxLoanAmount" size="11"
                value="150000"></td>
        <td id="cellTextScheduledPayment" width="200">Scheduled Payment</td>
        <td id="cellPmt" width="75"></td>
</tr>
        <tr id="T1Row2" >
        <td id="cellTextAnnualInterestRate" >Annual Interest Rate</td>
        <td id="cellTextboxIntRate"><input type="text" name="textboxIntRate" size="11"
value="6.5"></td>
        <td id="cellTextScheduledNumberof Payments">Scheduled Number of
Payments</td>
        <td id="cellNumPmtsSch"></td>
</tr>
<tr id="T1Row3" >
        <td id="cellTextLoanPeriodinYear">Loan Period in Years</td>
        <td id="cellTextboxLoanYears"><input type="text" name="textboxLoanYears"
        size="11" value="1"></td>
        <td id="cellTextActualNumberofPayments" >Actual Number of Payments</td>
        <td id="cellNumPmtsAct"></td>
```

```
</tr>
<tr id="T1Row4" >
        <td id="cellTextPaymentsPerYear">Payments Per Year</td>
        <td id="cellTextboxPaymentsPerYear"><input type="text"
        name="textboxPmtsPerYear" size="11" value="12"></td>
        <td id="cellTextTotalEarlyPayments" >Total Early Payments</td>
        <td id="cellErlyPmts"></td>
</tr>
<tr id="Row5" >
        <td id="cellTextLoanStartingDate" >Loan Starting Date</td>
        <td id="cellTextboxStartDate"><input type="text" name="textboxStartDate"
        size="11" value="01/01/2010"></td>
        <td id="cellTextTotalInterestPaid" >Total Interest Paid</font></td>
        <td id="cellIntPmts"></td>
</tr>
<tr id="Row6" >
        <td id="cellTextOptionalExtraPayments" >Optional Extra Payments</td>
        <td id="cellTextboxExtraPmts" ><input type="text"
name="textboxExtraPmts" size="11" value="0"></td>
        <td id="cellButtons" colspan="2">
        <input type="submit" value="Calculate Amortization"
        name="buttonAmortization" id = "buttonAmortization"
        onclick="showAmort()">
        <input type="submit" value="Reset" name="buttonReset" id = "buttonReset"
onclick="resetPage()">
        <input type="submit" value="Print Page" name="buttonPrint"
id="buttonPrint" onclick="printPage()"></td>
</tr>
</table><table border="1" width="80%" bordercolor="#800000"
id="tableAmortization">
<tr id="T2Row1" >
        <td id="cellTextPaymentNumber" width="50" align="center">Payment
Number</td>
        <td id="cellTextPaymentDate" width="90" align="center">Paymemt
Date</td>
        <td id="cellTextBeginningBalance" width="80" align="center">Beginning
Balance</td>
        <td id="cellTextScheduledPayment" width="70" align="center">Scheduled
Payment</td>
        <td id="cellTextExtraPayment" width="70" align="center">Extra
Payment</td>
        <td id="cellTextTotalPayment" width="70" align="center">Total
Payment</td>
        <td id="cellTextAmountToPrincipal" width="70" align="center">Amount to
Principal</td>
```

```
		<td id="cellTextAmountToInterest" width="70" align="center">Amount to
Interest</td>
		<td id="cellTextEndingBalance" width="80" align="center" >Ending
Balance</td>
		</tr></table>
</div>
</body>
<SCRIPT type="text/javascript" >
		function showAmort()
{
			document.getElementById("buttonAmortization").disabled=true;
			document.getElementById("buttonReset").disabled=false;
			document.getElementById("buttonPrint").disabled=false;
			var objExcel = new ActiveXObject("Excel.Application");
				objExcel.Visible = false;
				var excelPath ="C:/Program Files (x86)/Microsoft
		Office/Templates/1033/LoanAmortization.xltx";
				objExcel.Workbooks.open(excelPath);
				var excelApp = objExcel.ActiveSheet;
					excelApp.Cells(5,4).Value=
			textboxLoanAmount.value;
					excelApp.Cells(6,4).Value =
				textboxIntRate.value/100;
					excelApp.Cells(7,4).Value =
			textboxLoanYears.value;
					excelApp.Cells(8,4).Value =
			textboxPmtsPerYear.value;
					excelApp.Cells(9,4).Value =
			textboxStartDate.value;
					excelApp.Cells(10,4).Value =
			textboxExtraPmts.value;
				document.getElementById('cellPmt').innerHTML=
		"$"+excelApp.Cells(5,10).value.toFixed(2);
				document.getElementById('cellNumPmtsSch').innerHT
		ML=excelApp.Cells(6,10).value;
				document.getElementById('cellNumPmtsAct').innerHT
		ML=excelApp.Cells(7,10).value;
					document.getElementById('cellErlyPmts').inne
				rHTML="$"+excelApp.Cells(8,10).value.toFixed(2);
				document.getElementById('cellIntPmts').innerHTML="$
				"+excelApp.Cells(9,10).value.
				toFixed(2);
			var StartRow = 18
			var StartCol = 1
			var NumPmtsAct = excelApp.Cells(7,10).value;
			var myAmortArr = new Array(NumPmtsAct);
```

```
                var myReplaceArr = new
        Array("Mon","Tue","Wed","Thu","Fri","Sat","Sun","00:00:00","EDT","EST
        ");
        for (var j = 0; j< myAmortArr.length; j++)
        {
                var
x=document.getElementById('tableAmortization').insertRow(j+1);
                for (var i = 0; i<9; i++)
                {
                        var y=x.insertCell(i);

        myAmortArr[j]=excelApp.Cells(StartRow+j,StartCol+i).value;
                        if (i>1)
                        {
                                myAmortArr[j] = myAmortArr[j].toFixed(2);
                        }
                                y.innerHTML=myAmortArr[j];
                        if (i==1)
                {
                        for (var k = 0; k < myReplaceArr.length;k++)
                        {
                                y.innerHTML=y.innerHTML.replace(myRepla
                ceArr[k],"");
                        }
                }
        }
}
objExcel.ActiveWorkBook.Close(false);
objExcel.Quit();
}
</SCRIPT>
<SCRIPT type="text/javascript" >
        function versionCheck()
{
        document.getElementById("buttonAmortization").disabled=false;
        document.getElementById("buttonReset").disabled=true;
        document.getElementById("buttonPrint").disabled=true;
                var browserType=navigator.appName;
                var browserVersion=navigator.appVersion;
                var parseVersion=parseFloat(browserVersion);
                var objExcel;
        objExcel = new ActiveXObject("Excel.Application");
                var excelVersion = objExcel.version;
        objExcel.Quit();
        if (excelVersion < 12)
```

```
{
                alert("This Program Requires Excel 2007.  Code
                Modifications Required for Earlier Versions - Please Close
                                Program");
                        document.getElementById("buttonAmortization").
        disabled=true;
        }
        else if (browserType!="Microsoft Internet Explorer"||parseVersion <
4)
        {
                alert("This Program Requires MicroSoft I.E. 5.0 or Newer
Browser- Please Close Program");
                        document.getElementById("buttonAmortization").
        disabled=true;
        }
}
</SCRIPT>
<SCRIPT type="text/javascript">
        function resetPage()
{
        window.location.reload();
}
</SCRIPT>
<SCRIPT type="text/javascript">
        function printPage()
{
        window.print();
}
</SCRIPT>
</html>
```

Conclusions

Although there are few published examples available for programming within the HTA environment, this paper provides several examples of simple and more advanced HTAs using basic JavaScripting, FOS and COM technology. The interested reader is encouraged to visit the web sites referenced in this article as well as search the web for key words like “HTA example” and document relevant sights. Microsoft provides an extensive example of using HTA with COM and ActiveX Objects to build a web page editor (Introduction to HTML Applications (HTAs) (n.d.)).

References

COM Objects (n.d.). Microsoft® Windows® 2000 Scripting Guide. Retrieved from http://www.microsoft.com/technet/scriptcenter/guide/sas_vbs_wcmr.mspx?mfr=true

External Object (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/ms535246(VS.85).aspx

FileSystemObject Basics (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/d6dw7aeh(VS.85).aspx

FileSystemObject Sample Code (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/ebkhfaaz%28VS.85%29.aspx

HTML Applications Reference (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/ms536473(v=VS.85).aspx

HTML DOM (n.d.). W3Schools. Retrieved from http://www.w3schools.com/HTMLDOM/default.asp

IDispatch (n.d.). Wikipedia. Retrieved from http://en.wikipedia.org/wiki/IDispatch

IDispatch Interface [Automation], (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/ms221608.aspx

IntelliSense (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/hcw1s69b.aspx

Introduction to HTML Applications (HTAs) (n.d.). MSDN Library. Retrieved from http://msdn.microsoft.com/en-us/library/ms536496(VS.85).aspx

What is COM? (n.d.). Microsoft. Retrieved from http://www.microsoft.com/com/default.mspx

W3Schools (n.d.). W3Schools. Retrieved from http://www.w3schools.com/html/DEFAULT.asp

Appendix A

HTML Applications Reference (n.d.)

ELEMENT (tag)

- **<HTA: APPLICATION>** - Enables running of HTA.

ATTRIBUTES/PROPERTIES

- **APPLICATION -** Indicates whether the content of the object is an HTA, which is exempt from the browser security model.
- **applicationName -** Sets or gets the name of the HTA.
- **Border -** Sets or gets the type of window border for the HTA.
- **borderStyle -** Sets or gets the style set for the content border in the HTA window.
- **caption -** Sets or gets a Boolean value that indicates whether the window is set to display a title bar or a caption, for the HTA.
- **commandLine -** Gets the argument used to launch the HTA.
- **contextMenu -** Sets or gets a string value that indicates whether the context menu is displayed when the right mouse button is clicked.
- **Icon -** Sets or gets the name and location of the icon specified in the HTA.
- **innerBorder -** Sets or gets a string value that indicates whether the inside 3-D border is displayed.
- **maximizeButton -** Sets or gets a Boolean value that indicates whether a Maximize button is displayed in the title bar of the HTA window.
- **minimizeButton -** Sets or gets a Boolean value that indicates whether a Minimize button is displayed in the title bar of the HTA window.
- **navigable -** Sets or gets a string value that indicates whether linked documents are loaded in the main HTA window or in a new browser window.
- **Scroll -** Sets or gets a string value that indicates whether the scroll bars are displayed.
- **scrollFlat -** Sets or gets a string value that indicates whether the scroll bar is 3-D or flat.
- **Selection -** Sets or gets a string value that indicates whether the content can be selected with the mouse or keyboard.
- **showInTaskBar -** Sets or gets a value that indicates whether the HTA is displayed in the Microsoft Windows taskbar.
- **singleInstance -** Sets or gets a value that indicates whether only one instance of the specified HTA can run at a time.
- **sysMenu -** Sets or gets a Boolean value that indicates whether a system menu is displayed in the HTA.
- **Version -** Sets or gets the version number of the HTA.
- **windowState -** Sets or gets the initial size of the HTA window.